UNIVERSITAT POLITÈCNICA DE CATALUNYA

BACHELOR THESIS

Design of a Modular Exponentiation Module for an RSA Cryptographic Coprocessor with Power Analysis Countermeasures

Author:
Néstor TUNEU ARROYO

Director:
Dr. Álvaro GÓMEZ PAU

Co-Director:
Dr. Salvador MANICH BOU

Bachelor's degree in Industrial Technology Engineering
Grau en Enginyeria en Tecnologies Industrials

Escola Tècnica Superior d'Enginyeria Industrial de Barcelona (ETSEIB)

Department of Electronic Engineering

June 2018

Acknowledgements to my co-directors

Dr. Álvaro Gómez & Dr. Salvador Manich

for cheering me up and technical support

Abstract

Design of a modular exponentiation module for an RSA cryptographic coprocessor with power analysis countermeasures

by Néstor TUNEU ARROYO

Rivest-Shamir-Adleman (RSA) is a widely used public key cryptographic method. The main operation performed in this method, for encryption and decryption, is modular exponentiation. The way modular exponentiation is computed make the system vulnerable to side-channel attacks. Side-channel attacks focus on the physical implementation rather than in the algorithms vulnerabilities. In particular, power analysis attacks are a type of side-channel attack that focuses on extracting information from the power consumption trace.

The main thesis goals are to design, verify and obtain the specifications of a *Simple Power Analysis* (SPA) resistant coprocessor. A coprocessor and the hardware design are introduced because the case of study in this thesis requires a fast implementation of the RSA method.

The proposed design work with 4096-bit keys, following the recommendations of *NIST Special Publication 800-57 Part 1*. Thus, the design focuses on area optimization while dealing with large keys.

This design is presented in an easy-going schematic form, but, the fully functional version is presented using the hardware description language VHDL. By using *Cadence* ® software, the design is simulated and the implemented countermeasures are verified with a 16-bit version. These proposed countermeasures seek not to increase power consumption or execution time. In order to compare against an SPA vulnerable system, this reference version is also designed and simulated. The power traces for both versions are obtained to assess the effectiveness of the applied countermeasure.

In order to get realistic results, the design has been synthesized in a 1.2V standard 65 nm CMOS library.

The final proposed solution manages the area problem by using only one 4098-bit adder / subtractor into a *Montgomery Product* (MP) sequential scheme. This adder / subtractor is a type of *Parallel Prefix Adder* (PPA), in order to reduce delay. In particular, Ladner-Fischer topology is used. This reduces the number of wire tracks and logic levels, which help to synthesize this kind of huge adder.

The specifications obtained for the 4096-bit version allow the main system clock to run at about 100 MHz. In the SPA resistant version, this means a modular exponentiation can be computed, in average, in about 504 ms.

Resum

Disseny d'un mòdul d'exponenciació modular per a un co-processador criptogràfic RSA amb contramesures d'anàlisi de potència

per Néstor TUNEU ARROYO

Rivest-Shamir-Adleman (RSA) és un mètode criptogràfic de clau pública àmpliament utilitzat. La operació principal en aquest mètode, per encriptar i desencriptar, és l'exponencial modular. La manera com es computa l'exponencial modular fa el sistema vulnerable a atacs de canal lateral. Els atacs de canal lateral se centren en atacar el dispositiu físic, i no els algorismes. En particular, els atacs d'ànalisi de potència són un tipus d'atac de canal lateral que se centren en extreure informació de la traça de consum.

Els objectius principals del treball són dissenyar, verificar i obtenir les especificacions d'un co-processador resistent a atacs simples d'ànalisi de potència (SPA). El co-processador i el disseny en hardware s'introdueixen ja que el cas d'estudi del treball demanda una implementació ràpida del mètode RSA. El disseny proposat funciona amb claus de 4096 bits, seguint les recomanacions de *NIST Special Publication 800-57 Part 1*. Així doncs, el disseny se centra en optimitzar l'àrea, tot operant amb claus llargues.

El disseny es presenta de manera senzilla en format esquemàtic, però, la versió plenament funcional es presenta mitjançant codi VHDL. Mitjançant el programari de *Cadence* ®, el disseny és simulat i les contramesures verificades, amb una versió de 16 bits. Aquestes contramesures busquen no augmentar el consum de potència o el temps d'execució. Per tal de comparar aquest disseny amb un de vulnerable front SPA, aquest últim també és simulat. Les traces de consum per ambdues versions s'obtenen per tal d'avaluar l'efectivitat de la contramesura.

Per tal d'obtenir resultats realistes, la síntesi es realitza utilitzant una llibreria CMOS estàndard d'1.2V i 65 nm.

La solució final aborda el problema de l'àrea utilitzant únicament un sumador/restador de 4098 bits dins un esquema seqüencial d'un producte de *Montgomery* (MP). Aquest sumador /restador és del tipus prefix paral·lel (PPA), per tal de reduir el retard. En particular, s'ha utilitzat la topologia Ladner-Fisher. Això redueix el nombre de connexions i nivells lògics, cosa que ajuda en la síntesi d'un sumador tan gran. Les especificacions obtingudes per la versió de 4096 bits permeten que el rellotge principal del sistema pugui operar a uns 100 MHz. En la versió resistent a SPA, això vol dir que l'exponencial modular pot ser calculada, en mitjana, en uns 504 ms.

ETSEIB

Contents

List of Figures

List of Tables

List of Abbreviations

AES	**A**dvanced **E**ncryption **S**tandard
CLA	**C**arry **L**ook-ahead **A**dder
CMOS	**C**omplementary **M**etal **O**xyde **S**emiconductor
DP	**D**ot **P**rocedure
DPA	**D**ifferential **P**ower **A**nalysis
ECC	**E**lliptic **C**urve **C**riptography
HO-DPA	**H**igh **O**rder **DPA**
IP	**I**terative **P**rocess
MP	**M**ontgomery **P**roduct
NRE	**N**on-**R**ecurring **E**ngineering
OAEP	**O**ptimal **A**symmetric **E**ncryption **P**adding
PKCS	**P**ublic-**K**ey **C**ryptography **S**tandard
POS	**P**roduct **O**f **S**ums
PPA	**P**arallel **P**refix **A**dder
RSA	**R**ivest **S**hamir **A**dleman
SOP	**S**um **O**f **P**roducts
SPA	**S**imple **P**ower **A**nalysis
VHDL	**VHSIC** **H**ardware **D**escription **C**ode
VHSIC	**V**ery **H**igh **S**peed **I**ntegrated **C**ircuit

List of Symbols

#	number of
α	exponent
β, β_t	base (MP variable change)
π	array of g,p or G,P pairs
a, b	operand (addition)
c	carry (addition)
C_{12}	counter's twelfth output
CLK	clock signal
e, e_t	modular exponentiation (MP variable change)
E	encrypted message
EN	enable signal
f	variable (Algorithm 2.4)
g	generate function
G	generalized generate function
h	partial result (Multiply and Reduce)
i, j, k	iterator
K	private key
m, M	decrypted/plaintext message
n	number of bits (generic)
p	propagate function
P	generalized propagate function
q	quotient
Q	public key
r	remainder
s	result (addition)
S	selection signal
t	delay (time)
u	group size
v, w	prime number
x, y	operand (product)
z	auxiliar logic function

Chapter 1

Introduction

1.1 Personal Background

As a student, in the last year, I have been working with *Elliptic Curve Cryptographic* (ECC) systems in the QiNE research group at the ETSEIB-UPC's Department of Electronic Engineering. The work have been about differential scan-based side-channel attacks and countermeasures in cryptographic systems designed for testability. Since I had to do some bibliographic search in that topic, I realized there are other widely used public key methods.

That is the reason why I have choosen *Rivest, Shamir* and *Adleman* (RSA) method in this work, because it is the other main asymmetric cryptographic system used nowadays, which I did not know nothing about. In the same way, I would like to explore other kinds of attacks and countermeasures, different from differential scan-path. Power analysis techniques play that role in this thesis.

Also, I am interested in high performance arithmetic circuits. Few months ago, I started reading (by own decision) the book *Synthesis of Arithmetic circuits: FPGA, ASIC and Embedded Systems*, which also is the main bibliographic reference I have used [1]. The merge of these facts is my main motivation to work on.

1.2 Case of Study

Despite main topics in this work have not been presented yet, with a few definitions the case of study can be presented. A public key cryptographic system is a communication system that allows to send and receive messages through an insecure channel. Thus, an external opponent would not be able to get information without the hidden private key, which is a long prime number. Encrypting operation is not bijective or is too much complex. Then, even though public key can be obtained through the insecure channel, there is not enough information to decrypt the messages. Nowadays, for example, all this is important to increase security on mobile instantaneous wireless communications.

In general, RSA, ECC and other cryptographic systems use that kind of long private keys. Key length is related to security in these systems, but it is a challenge for the system performing the computation. This is because execution time is related in some way with the length of the key and messages. Without a dedicated hardware, to execute cryptographic algorithms in simple processors could be too slow for some applications. Communication delay, as shown in the mobile example, it is an important factor in real time systems.

Let us suppose in this work that the final application requires a fast implementation, thus introducing the necessity of a cryptographic coprocessor and discarding slower software implementations on microprocessors.

Then, two more important facts appear in the design stage. Firstly, the area. Area in electronics refeer to the physical surface where an integrated circuit is fabricated. Since keys and messages are long, circuits to operate with them are going to be large too. The way this issue has been managed in this work is to reuse some physical circuits in different algorithm operations of the same kind, which is the same as sequentializing. This technique will probably slow down the execution, so fast arithmetic circuits are required.

Secondly, the power consumption. Power consumption depends on the circuit architecture and transistor technology. Since the architecture variables will be fixed with delay and area constrains, and technology is a too wide topic to consider with rigor in this work, the power consumption will be considered a dependent variable. Nevertheless, the power consumption will be important when designing against power analysis attacks.

Power analysis is a type of side-channel attack. Since cryptographic methods are designed to be resistant to algorithm attacks, side-channel attacks are a type of attack that focus on the physical implementation of the system, in order to retrieve the private key. In particular, power analysis focuses in the system power trace when an operation related with the private key is performed.

Thus, in order to find a solution to this case of study, thesis goals are:

- To elaborate a state of the art revision of arithmetic and power analysis attacks and countermeasures applied to RSA systems.
- To design a functional exponentiation module to encrypt or decrypt messages.
- To protect the designed module against *Simple Power Analysis* (SPA) attacks.

The way these goals are achieved through this thesis is: state of the art revision in *Chapter 2*, modular exponentiation module design and countermeasure selection in *Chapter 3*, verification plan and results presentation in *Chapter 4* and finally results discussion, costs estimation and conclusions in *Chapter 5*. *Appendix A* contains the complete VHDL module description. *Appendix B* contains *Cadence* ® the commands needed to carry out the simulations and the synthesis.

Chapter 2

State of the Art

This chapter includes an state of the art revision of the main topics of the work. Some definitions and algorithms are introduced, as well as the explanation for the RSA cryptographic method and power analysis attacks and countermeasures.

2.1 RSA Cryptographic Method

The first time RSA cryptographic method appeared was in *A method for obtaining digital signatures and public-key cryptosystems* [2] in 1978. Since then, there have not been changes in the structure of the method. Taking this into account, let us explain the method from an easy-going point of view. All the demonstrations and properties are included in [2]. Later, an overall scheme of an asymmetric cryptographic system will be introduced.

As mentioned in *Chapter 1*, RSA, as a secure communication method, allows the transmission of encrypted data through an insecure channel. To establish the communication, a public key is used. This public key has a relation with the private key, which is hidden by the owner. To illustrate how the process works, let us see an example:

The user A wants to send a message to the user B. Firsly, A sends a request to B to start the communication. B generates a public key and sends it to A. The generation of the public and private keys for user B is done using two prime numbers v and w. A public key Q is choosen (see numeric example) and the private key K is determined by usign this congruence:

$$Q \cdot K \equiv 1 \pmod{(v-1)\cdot(w-1)} \tag{2.1}$$

Later, A converts its message M using a padding scheme into a number m lower than $u = v \cdot w$. When this conversion is completed, A encrypts its message m by using the next equation, where E is the encryption of message m:

$$E = m^Q \pmod{u} \tag{2.2}$$

After that, A sends this information to B who decrypts the message by using the next formula and later M from m:

$$m = E^K \pmod{u} \tag{2.3}$$

As shown, there is only one operation needed to do all the cryptographic computations: modular exponentiation. It is for this reason that work focuses on that module. The padding scheme is used to get a transferable representation and to increase security. In the *Public-Key Cryptography Standard #1* (PKCS #1) the padding scheme is the *Optimal Asymmetric Encryption Padding* (OAEP). To illustrate the communication between the users, *Figure 2.1* shows the scheme of how transmission is done.

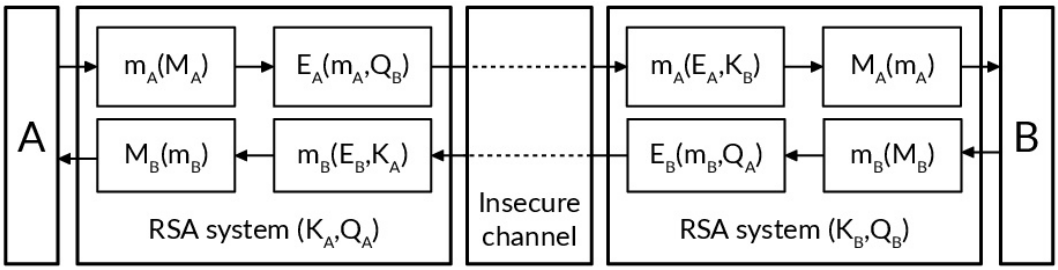

FIGURE 2.1: Public-key cryptograhic communication system

Let us see a numeric example. Let us suppose $v = 113$, $w = 221$ and $m = 18593$ after the padding. Then, $u = v \cdot w = 23843$. Now, Q is choosen as 37 (coprime with $\text{lcm}((v-1) \cdot (w-1)) = 6160$). By using (2.1), then $K = 333$. Thus, encryption and decryption in an A to B communication:

$$\text{(A) } E_A = {m_A}^{Q_B} \pmod{u} = 18593^{37} \pmod{23843} = 8010$$

$$\text{(B) } m_A = {E_A}^{K_B} \pmod{u} = 8010^{333} \pmod{23843} = 18593$$

One fundamental property of RSA is asymmetry. This concept applied to cryptography means two devices can communicate using two different private keys. As mentioned, they will need public key generations to set a secure communication. Usually, asymmetric methods are used to start symmetric key implementations. At the beginning, RSA can be used to secure send a common private key to start a symmetric communication method, which is faster.

The RSA method has been proven as completely secure if anyone can extract information from the physical devices (black box assumption), since private keys are stored within the system itself.

In the last decade cryptography has gotten relevance and the main differences between devices are the cryptographic operations and algorithms to compute modular exponentiation.

2.2 Power Analysis on RSA

Power analysis is a kind of side-channel attack, which focuses on the physical implementation of a system, rather than in its algorithms. Although there are more methods of power analysis than the two presented in this section, as HO-DPA (High Order), simple and differential power analysis are widely used because system security usually can be overpassed with them. The main reference used in this section is doctoral thesis [3].

2.2.1 Simple Power Analysis (SPA)

An SPA attack consists in analyzing the power consumption of a physical device when an operation using the private key is being performed. Usually devices perform different operations depending on the current bit of the private key. Many algorithms evolve sequentially using the private key bits. Therefore, the opponent can discover if the current bit in the private key is 0 or 1, by observing the power consumption trace.

In RSA, SPA can be used during a modular exponentiation. Depending on the bit of the private key, system performs a multiplication and a doubling when the bit is 1 and just doubling when the bit is 0. Consequently, the power consumption of the system is higher if the bit is 1.

There are three main ways to protect cryptographic systems against SPA. All respective original documents of the algorithms mentioned can be found in [3]. The first method, consists in performing the two different operations (multiplication and doubling) at the same time, either if one of them is unnecessary. This idea is the base of *Montgomery ladder algorithm*.

The second method consists in unifying the execution. This means to execute the same set of operations, without differences. An example of that is *Binary Huff curve-based SPA-resistant implementation*, in this case for ECC.

The last method consists in an atomic execution. A sequence of operations is stablished and always is done in the same way. An example of that is *Atomic right-to-left algorithm* for RSA.

Finally, although these methods can protect against SPA, they do not change the algorithm structure, data or partial results. This means all that the presented countermeasures can be overpassed with DPA.

2.2.2 Differential Power Analysis (DPA)

DPA, in opposition to SPA, do not observe the power consumption of different operations. DPA consists in observing the power consumption of different data through the same operation. However, the differences in power consumption on different data are tiny, so several

runs must be performed with a posterior offline statistical data analysis to retrieve the private key from the deduction of the partial results.

This kind of attacks are more complex, but they can deal with SPA countermeasures. Then, DPA countermeasures must be introduced. Most countermeasures are based on the idea of random masking. At the beginning of the algorithm, data is randomly masked. Then, operations are executed and finally the data is unmasked to retrieve the final result. On RSA, two types of data can be masked: the base or the exponent, both on modular exponentiation.

In [3], a DPA side-channel attack is presented to retrieve the private key successfully in an SPA countermeasures scenario. However, the same DPA attack becomes useless when DPA countermeasures are in place.

Once attacks and countermeasures have been introduced, it must be noticed that this thesis focuses on SPA attacks and countermeasures, as it has been presented in the thesis goals.

2.3 Fast Arithmetic on RSA

The aim of this section is to find an efficient algorithm for modular exponentiation. Since this algorithm will be sequentialized, also it is necessary to find a fast addition algorithm to allow an increase in the clock speed. The main reference in this section is *Synthesis of Arithmetic circuits: FPGA...* [1]. In particular, the chapters 3 (addition) and 8 (finite field). Support references are [4]–[6].

2.3.1 Modular Exponentiation

As seen in the RSA section, the main operation in RSA is modular exponentiation. Let us define a base β and an exponent α with the typical binary representation, where n is the number of bits and $\alpha_i \in [0,1]$:

$$\alpha = \alpha_{n-1} \cdot 2^{n-1} + \alpha_{n-2} \cdot 2^{n-2} + ... + \alpha_1 \cdot 2^1 + \alpha_0 \tag{2.4}$$

Then, the *Horner*'s scheme brings an efficient way to write the modular exponentiation $e = \beta^\alpha \pmod u$, where $u < 2^n$:

$$e = \beta^\alpha \pmod u = ((...((1^2 \cdot (\beta^{\alpha_{n-1}})^2) \cdot \beta^{\alpha_{n-2}})^2...)^2 \cdot \beta^{\alpha_1})^2 \cdot \beta^{\alpha_0} \pmod u \tag{2.5}$$

This structure, as an algorithm, includes a modular doubling operation for each bit and also includes a modular multiplication if $\alpha_i = 1$. In a modular doubling operation, e variable gets the remainder of the division $(e \cdot e)/u$, which also includes a multiplication. For each $\alpha_i = 1$, also one $(e \cdot \beta)/u$ division is performed:

Algorithm 2.1

$e = 1$

for $i = 1 : n$

$\quad e = e \cdot e \pmod{u}$

$\quad$ if $\alpha_i == 1 \rightarrow e = e \cdot \beta \pmod{u}$

end

As seen, there are too many operations per bit. To reduce the algorithm delay, Montgomery Product (MP) is introduced. Although MP is going to be explained later, below there is an algorithm which computes the modular exponentiation using it:

Algorithm 2.2

$e_t = 2^n \pmod{u}$

$\beta_t = MP(\beta, 2^{2n} \pmod{u})$

for $i = 1 : n$

$\quad e_t = MP(e_t, e_t)$

$\quad$ if $\alpha_i == 1 \rightarrow e_t = MP(e_t, \beta_t)$

end

$e = MP(e_t, 1)$

In the algorithm above, no divisions are performed, since mod u only affects constant values ($2^n \pmod{u}$ and $2^{2n} \pmod{u}$), that can be computed previously.

2.3.2 Modular Product

As introduced in the last subsection, MP reduces the delay in the modular exponentiation. Thus, let us explain why by analyzing the modular product.

In *Algorithm 2.1* the main operation is the modular product. A first way to calculate it is the *Multiply and reduce* scheme, where $x \cdot y$ product is computed to obtain h and then a division is performed to compute the quotient q and the remainder r, which is the result of a modulo u operation.

Since division is a long operation, an alternative to *Multiply and reduce* is performing the modulo operation at every step. This strategy is known as *Modulo u Shift-and-add algorithm*. *Algorithm 2.3* is the general base-2 version, that is named *Shift-and-add* because multiplication by 2 is left-shift in binary.

Algorithm 2.3

$h_n = 0$

for $i = 0 : n - 1$

$$h_{n-1-i} = (2 \cdot h_{n-i} + x_{n-1-i} \cdot y) \pmod{u})$$

end

$r = h_0$

By using some properties, the last algorithm can be simplified to avoid modular operations. Since this algorithm is not going to be used, it is not worth it to explain it with detail. However, the idea is also used in MP: to avoid division operation to obtain remainder. Let us see how this can be achieved.

Given an odd value u, in particular, prime, and the operands $x < u$, $y < u$, the idea is try to find a number r' so that:

$$x \cdot y \pmod{u} = r' \cdot 2^n \pmod{u} \tag{2.6}$$

Since u is odd, the greatest common divisor between u and 2^n is 1. Then, it has to exist a number 2^{-n} so that $2^n \cdot 2^{-n} = 1$. Thus:

$$r' = x \cdot y \cdot 2^{-n} \pmod{u} \tag{2.7}$$

This equation is used in MP to compute r', shown below. Lemmas 8.1 and 8.2 in [1] include an explanation of why f is defined in this way and a proof of $r_n < 2u$, which means r' is r_n or $r_n - u$.

Algorithm 2.4

$r_0 = 0$

for $i = 1 : n$

$$f = r_{i-1} + x_{i-1} \cdot y$$

$$r_i = (f + f_0 \cdot u)/2$$

end

if $r_n < u \rightarrow r' = r_n$

if $r_n \geq u \rightarrow r' = r_n - u$

Once algorithm is defined, let us explain how modular product is computed from MP. By using expression (2.7), the next relations can be deduced:

$$r = x \cdot y \pmod{u} = x \cdot y \cdot 2^{-n} \cdot 2^{2n} \cdot 2^{-n} \pmod{u} = r' \cdot 2^{2n} \cdot 2^{-n} \pmod{u} \tag{2.8}$$

$$r = MP(MP(x,y), 2^{2n} \pmod{u}) \tag{2.9}$$

The expression (2.9) also defines the algorithm to compute modular product. In this relation $2^{2n} \pmod{u}$ must be precomputed to reduce the execution time. In *Algorithm 2.2* a change of variable is performed to simplify computation and at the end of the algorithm it is inverted by using:

$$MP(e_t, 1) = e_t \cdot 2^{-n} \pmod{u} = e \cdot 2^n \cdot 2^{-n} \pmod{u} = e \pmod{u} \tag{2.10}$$

Since *Algorithm 2.4* is composed of additions and subtractions, the next subsection will focus on such kind of arithmetic operations and how they can be efficiently implemented in hardware.

2.3.3 Addition / Subtraction

To begin with the addition review, let us present in the first place the handwritten binary addition algorithm. In *Algorithm 2.5*, a and b represent the operands, c the carry and s the result. This type of adder is known as *Ripple Carry Adder*.

Algorithm 2.5

---

for $i = 0 : n-1$

$s_i = a_i + b_i + c_i \pmod{2}$

if $a_i + b_i + c_i > 1 \rightarrow c_{i+1} = 1$

if $a_i + b_i + c_i \leq 1 \rightarrow c_{i+1} = 0$

end

Notice that, the algorithm execution time is proportional to the number of bits n. This delay can be reduced by defining two new functions: propagate carry p_i and generate carry g_i. Explained in words, generation happens when the input c_i does

not matter and the carry output c_{i+1} is 1. This is equivalent to $a_i + b_i = 2$ or logic operation $g_i = a_i \; and \; b_i$.

Propagation happens when $a_i + b_i = 1$, which is equivalent to the logic operation $p_i = a_i \; or \; b_i$. Propagation implies that $c_{i+1} = c_i$. Since propagation and generation only depend on operands, they can be computed in parallel. *Algorithm 2.6* has an execution time presenting a linear dependence with n, but not strictly proportional.

Algorithm 2.6

for $i = 0 : n - 1$

$p_i = a_i$ or b_i

$g_i = a_i$ and b_i

end

for $i = 0 : n - 1$

$c_{i+1} = g_i$ or $(p_i$ and $c_i)$

end

for $i = 0 : n - 1$

$s_i = a_i + b_i + c_i \pmod 2$

end

In fact, a last change can be done to achive the fastest adder, known as *Carry Look-ahead Adder* (CLA). As seen, the carry function is:

$$c_{i+1} = g_i \text{ or } (p_i \text{ and } c_i) = g_i + p_i c_i \tag{2.11}$$

Thus, by using a composition, carry computation loop can almost be parallelized to achieve logaritmic delay in relation to n. *Algorithm 2.7* corresponds to the idea of a n-bit CLA adder where each loop can be parallelized, as carry computations. The notation has changed: + means *or*, $\oplus$ means *xor* and $\cdot$ means *and*.

Algorithm 2.7

for $i = 0 : n - 1$

$p_i = a_i + b_i$

$g_i = a_i b_i$

end

$c_1 = g_0 + p_0 c_0$

$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$

$c_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$

...

$c_n = g_{n-1} + p_{n-1} g_{n-2} + p_{n-1} p_{n-2} g_{n-3} + ... + p_{n-1} p_{n-2} ... p_1 p_0 c_0$

for $i = 0 : n - 1$

$s_i = a_i \oplus b_i \oplus c_i$

end

In the last algorithm, carry computation pattern is shown. Anyway, if n is large, specific algorithms are defined to compute the recurrent function of the carries. This type of adders are known as *Parallel Prefix Adders* (PPA). The proposed design is going to be one of this kind.

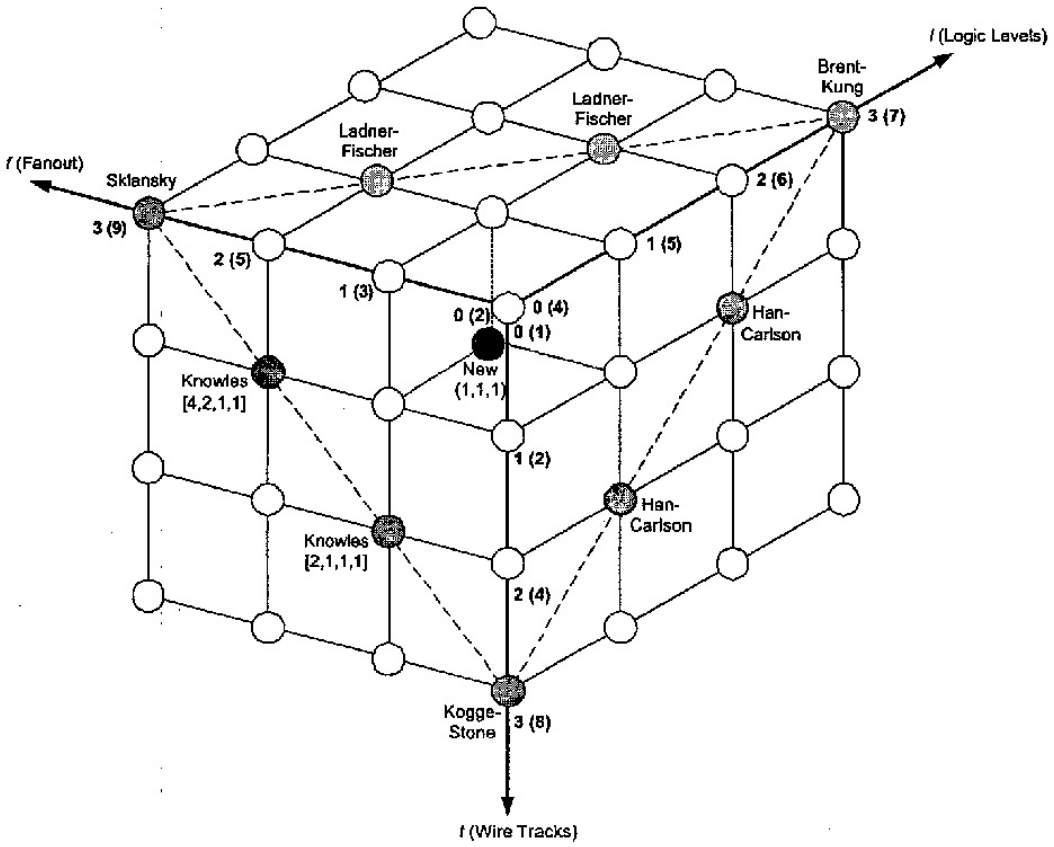

FIGURE 2.2: Taxonomy of prefix graphs [7]

In the 2003 paper *A taxonomy of parallel prefix networks* [7], an excellent PPAs compilation can be found. In fact, there is a great 3D comparative analysis of the most known PPA topologies which supports the election of the best topology for each application. *Figure 2.2* shows this 3D graph comparative.

In [1], Brent-Kung and Ladner-Fischer topologies are explained in detail. Nowadays, the Kogge-Stone topology is the fastest. However, as seen in the graph, its high number of wire tracks makes a 4098-bit synthesis unappropiate using it. In the design chapter, the selection of the topology will be discussed.

As commented in the last section, also a subtraction is needed in the MP. In *Algorithm 2.7*, the subtraction is computed by making the logical inverse of b and by setting $c_0 = 1$. Therefore, c_0 can be used as an add/subtract selection signal. This is known as *Two complement addition*, that is the same as subtraction.

To conclude this section, remember that CLA is introduced because in RSA cryptography, the number of bits n is large, and a proportional or linear delay would make the execution too slow.

2.4 CMOS and Other Considerations

To conclude this chapter, we are going to talk about the implications of manufacturing technology. In order to properly design a modular exponentiation module, gates non idealities must be taken into account. Two of these non idealities are gate delay and fanout. The information of this subsection has been obtained from [8].

Complementary Metal Oxyde Semiconductor (CMOS) is a manufacturing technology which uses pairs of nNOS and pMOS transistors forming a p network in charge of driving the output to the logic level 1 and a n network in charge of driving the output to the logic level 0. In few words, the advantages are a low static power consumption and a great robustness against noise. CMOS is the mainly used fabrication technology nowadays, that is the reason why it has been choosen for this work.

In this work, three parameters are considered crucial for the targeted application: number of transistors and gate delay and fanout. In CMOS technology, NOT gates have one nMOS and one pMOS transistors, 2-input NOR and NAND gates have two of each kind. XOR, AND and OR gates are compositions of the previous gates.

However, although NOR and NAND gates have the same number of transistors, the NAND gate presents a lower delay. This is because a NAND has two series nMOS transistors and a NOR has two series pMOS. Since pMOS transistors are slower (less

conductive) than nMOS ones, the NAND gate is faster for equivalent dimensioned transistors.

This is an important fact to be considered in order to design a fast circuit with a large number of gates. In fact, since NAND gate is a complete set of gates, any logic function can be designed using it.

The fanout concept refeers to the number of gates that can be linked to an output line of a gate. In the library used in this thesis, for example, the 3-input NOR gate with 2-fanout is named NOR3X2.

Once manufacturing technology implications have been explained, let us introduce other considerations:

- *Quine-McCluskey algorithm* and *Karnaugh map* are two of the techniques which reduce logic expressions to their minimal form. The two ways to express that form are *Sum of Products* (SOP) and *Product of Sums* (POS), where product and sum are logic operations. These algorithms are going to be required to present the schematics in the design chapter.
- *Hardware Description Languages* (HDLs) are languages used to define circuits on behaviour or structural domain. There are compilers which interpret HDLs to synthesize electronic circuits. Then, these circuits can be simulated or fabricated. The main HDLs are *VHSIC* (Very High Speed Integrated Circuit) HDL (VHDL) and *Verilog*.
- The software used in this work includes *ncsim*, *ncelab*, *ncvhdl* and *Encounter RTL Compiler*. These are all *Cadence* ® software required to simulate and synthesize the VHDL based designs.

The next chapter is dedicated specifically to the design and the implementation of the modular exponentiation module with security countermeasures.

Chapter 3

Secure Modular Exponentiation Module Design

This chapter includes the design of the modular exponentiation module. When necessary, some algorithms and schematics are also included, some of them as an extension of the state of the art chapter.

3.1 Arithmetic Approach

In order to start designing the modular exponentiation module, this section introduces its arithmetic circuits in structural domain, each one linked to the algorithms presented in the last chapter.

3.1.1 Overall Structure

The first main decision that has to be taken to define the structure is the private key size. In *NIST Special Publication 800-57 Part 1* [9], a recommendation of key size is given. Beyond 2030, for RSA cryptosystems, using private keys of 2048 bits or less will be not secure enough. So, considering that the system will be implemented in hardware with a long-term usage, a convenient key size for the design is 4096 bits.

Taking this into account, let us review *Algorithm 2.2* to start with the design. This algorithm has four steps. The first step consists in two changes of variable. One of them perform an MP. The second step is related with $MP(e_t, e_t)$. The third step computes $MP(e_t, \beta)$ and the last step is to undo the change of variable by using $MP(e_t, 1)$. Thus, these steps can be encoded in binary, using multiplexation to select each one of the steps. By doing that, the circuit is only going to need one MP operator. The next shematic, *Figure 3.1*, shows the overall circuit.

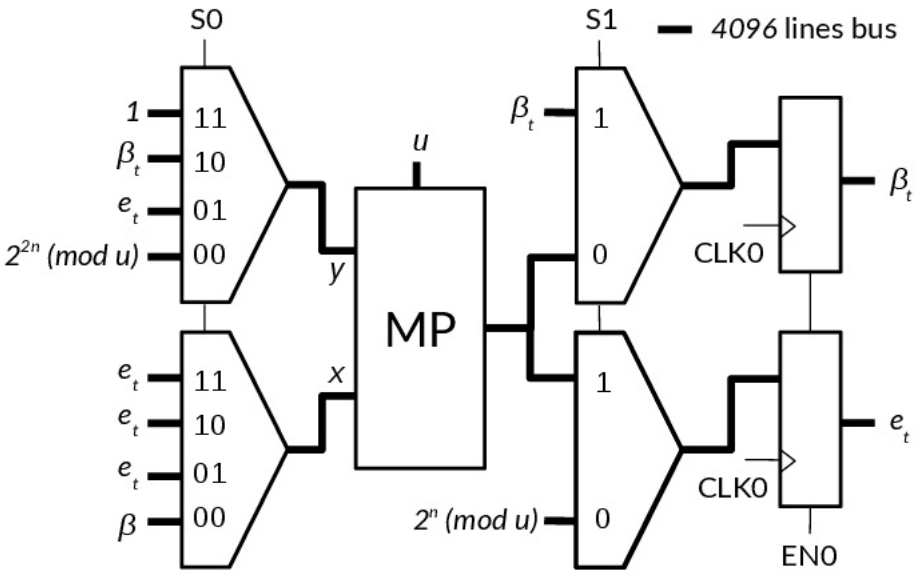

FIGURE 3.1: Overall schematic

As seen, D-type flip-flops have been included to allow sequential operation. When computation ends, e_t signal also contains the modular exponentiation result, as the last step (encoded as 11 in the multiplexers) undoes the change of variable.

S0 and S1 signals (S of select) have to be described with *Moore*'s state machines to obtain a correct functionality. Signal CLK0 (CLK of clock) corresponds to the overall circuit clock. Anyway, this clock is not going to be the slowest signal in the system because the signal S0 is going to require an end flag signal to reach 11 state. Then, let us define S0 and S1 signals and discuss about the enable signal EN0.

It must be noticed that all the schematics in this section are only orientative. *Encounter RTL Compiler* may synthesize the design in a different way. In this sense, and only to simplify schematics, flip-flops appear as synchronous with each CLK signal. In the VHDL description all flip-flops are synchronous to the fastest clock CLK2 and the other clocks are enable signals for each one. Also, there is an overall reset signal.

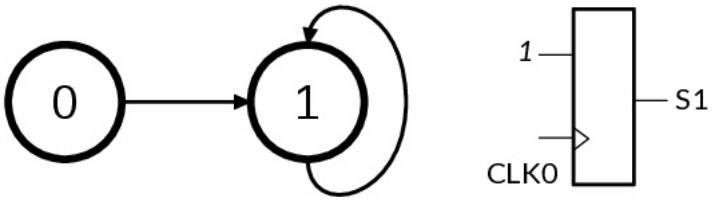

FIGURE 3.2: S1 signal generator

Since S1 is easier to define, let us start with it. S1 must be at logic 0 in the first clock and later at 1. This is a trivial state machine represented in *Figure 3.2*, where the names of the states correspond to the S1 output. It must be noticed that in this design, flip-flops requiring it, are initialized at logic 0 using the already mentioned reset signal, or any other clock if it is required.

The selection signal S0 is harder to define. First of all, it must be remembered that S0, which is the main selection signal, switches between doubling and multiplication. So, one important parameter is the current exponent bit α_i. Another parameter is an end flag which notes the end of the exponent. This allows to the circuit to undo the change of variable correctly.

The parameter, noted as C_{12}, can be obtained with a 13-bit standard counter working with $2 \cdot \text{CLK0}$. In fact, $2^{12} = 4096$, so the thirteenth output in counter is the required signal. As with S1, *Figure 3.3* shows the corresponding state machine and schematic:

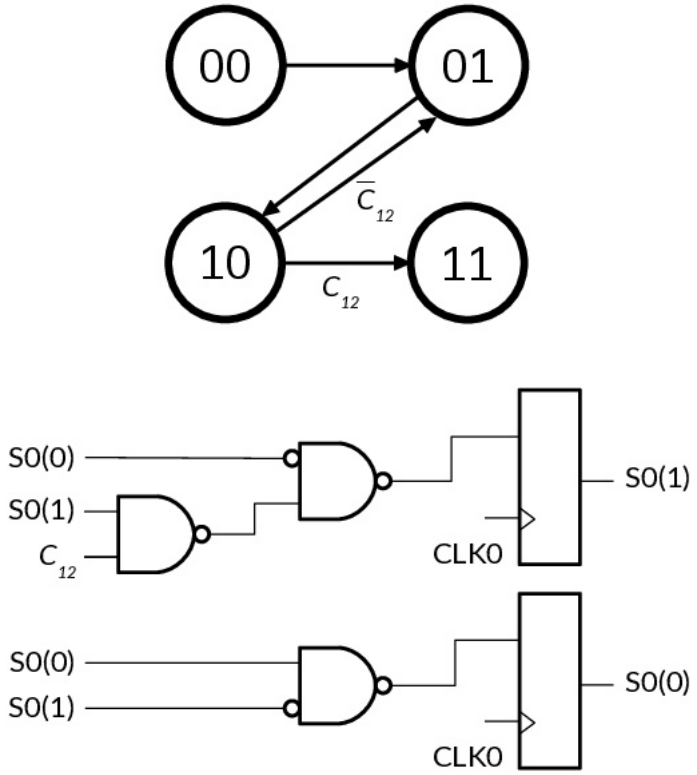

FIGURE 3.3: S0 signal generator

When $\alpha_i = 0$, only a doubling must be performed. This means EN0 = 0 after the doubling operation. Else, when $\alpha_i = 1$, EN0 = 1. Enable signal EN0 also affects the internal MP D-type flip-flops to avoid repeating another doubling operation.

Value α_i can be obtained from a shift register with α previously loaded and also working with $2 \cdot$ CLK0. The proposed SPA countermeasure is based in the timing control of this shift register and S0, but this will be introduced in the next section.

3.1.2 Montgomery Product Structure

MP operator structure can be deduced from *Algorithm 2.4*. In this case, let us analize the circuit in two steps. The first step consists in a global view of the multiplier without describing algorithm loop (Iterative Process, IP). This is shown in *Figure 3.5*.

As seen, the algorithm comparison is simplified by using the subtraction result sign, s_{4097}. To perform an MP, a 4098-bit adder and subtractor are required. Then, *Figure 3.4* shows the IP block internally. This circuit has been designed to use only one adder.

By taking a closer look on the two schematics, it can be seen that the adder and the subtractor can be replaced by only one operator, mixing the circuits, as introduced in *Chapter 2*. The result is shown in *Figure 3.6*.

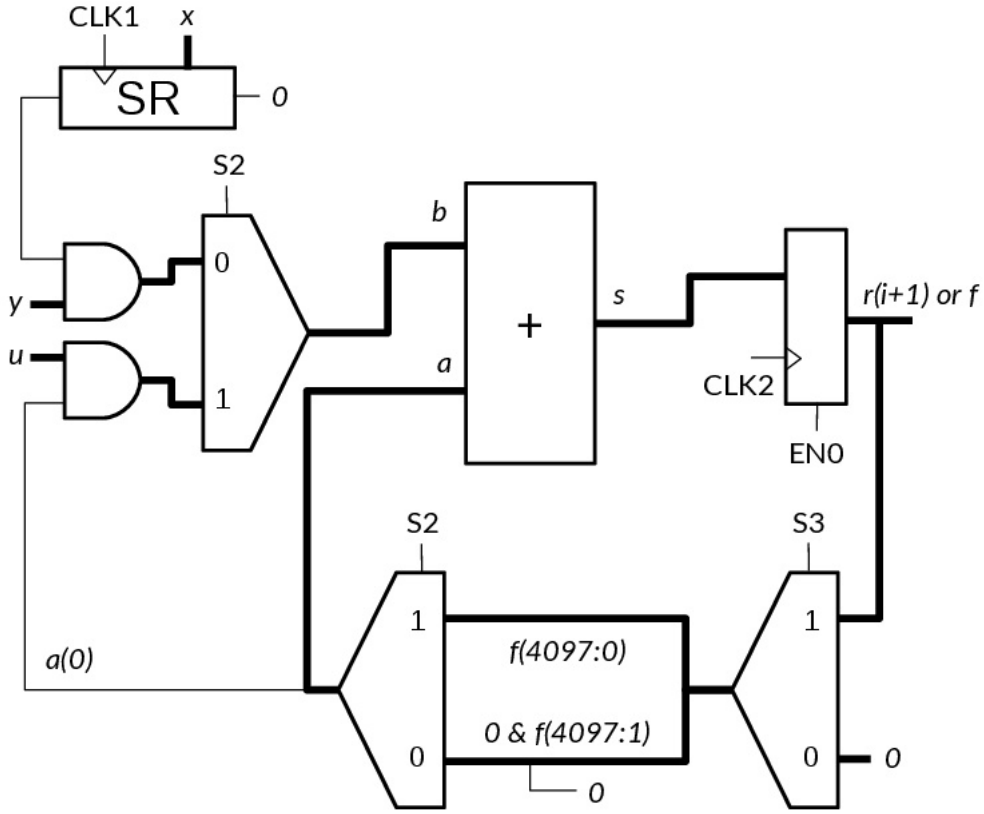

FIGURE 3.4: Iterative process schematic

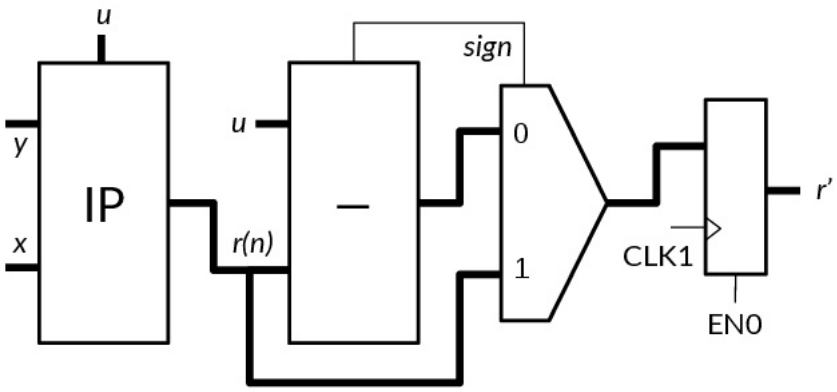

FIGURE 3.5: Montgomery Product partial schematic

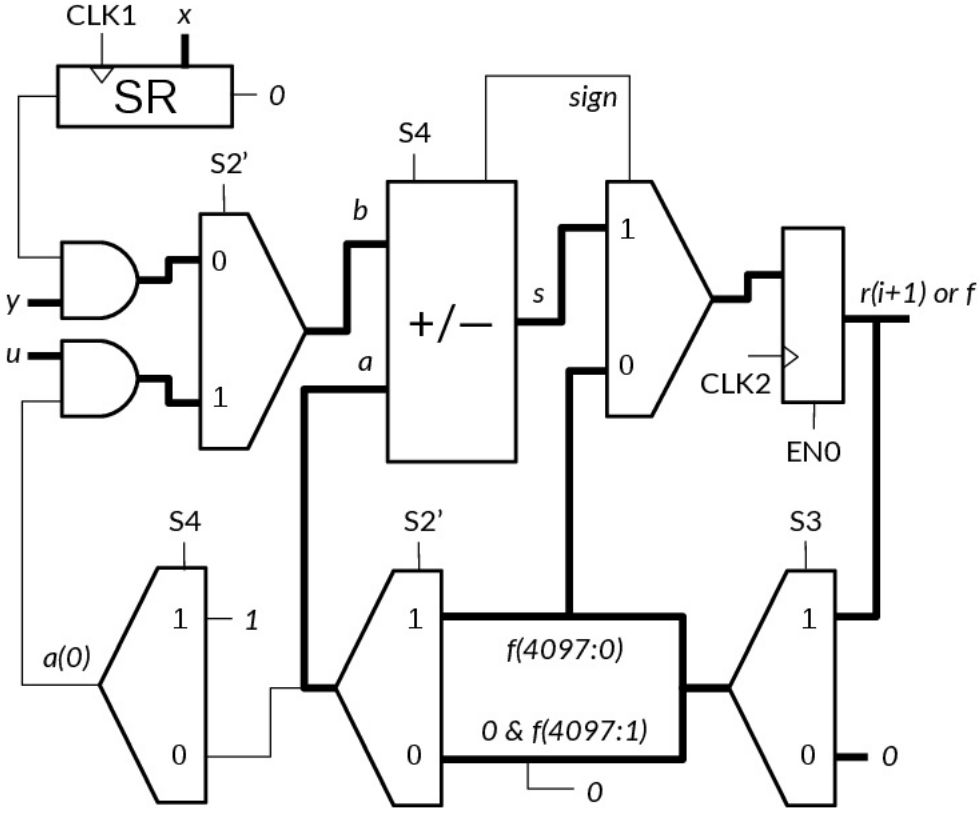

FIGURE 3.6: Montgomery Product schematic

Similarly to the previous subsection, let us design S2', S3 and S4 signal generators. S3 signal is equivalent to S1, but it has to been reset for each MP process (with CLK0). In *Figure 3.4*, S2 signal should alternate logic values 0 and 1. However, in *Figure 3.6*, the introduced modifications require a new signal S2' that works as S2 but after 4096 clocks it must stay at logic 1 to allow *u* subtraction. In fact, that end flag

signal is done, as C_{12}, with a counter, but this new counter C'_{12} must be reset with CLK0. Also, S2' can be expressed as S2' = S2 or C'_{12}, where S2 takes its inverse for each CLK2. Finally, S4 signal must take logic 1 after 4096 clocks, so it is equivalent to C'_{12}.

To wrap things up, the design shown in this subsection fulfills one of the goals introduced at the beginning: with only one adder/subtractor and a proper surrounding logic, modular exponentiation can be computed. This is a great enough area reduction by sequentializing.

3.1.3 Adder/Subtractor Structure

As shown in the state of the art chapter, the CLA is the fastest known adder when the number of bits is large. Also, this adder can be modified to operate as a subtractor just setting $c_0 = 1$ and inverting the subtrahend.

Since r_n has one more bit than the MP operands, the CLA must operate as a $(n+2)$-bit adder to be able to detect the result sign. Thus, as a 4098-bit CLA can not be shown in schematic form, in this section the design is going to be mostly treated as an algorithm.

To be coherent with the previous sections, this adder has to be designed to have a low delay, because it is the only large circuit which is combinational in the entire design. This is going to require some changes and extensions in the algorithms presented in the state of the art.

First of all, propagation function must be redefined in a more convenient form using *De Morgan*'s laws:

$$p_i = a_i + b_i = \overline{\overline{a_i} \cdot \overline{b_i}} \tag{3.1}$$

$$c_{i+1} = g_i + p_i c_i = \overline{\overline{g_i} \cdot \overline{p_i c_i}} \tag{3.2}$$

Since to operate as a subtractor the CLA requires inverters in the operands, expressing p_i as shown does not increase delay. In fact, delay is reduced because of using NAND gates. So, as shown in *Figure 3.7*, this circuit only uses NAND gates.

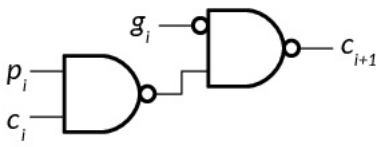

FIGURE 3.7: c_{i+1} computation

However, if these circuits are concatenated, a linear delay is expected. So, by using PPA algorithms, a logarithmic delay is going to be obtained. The question is which topology is the best for this application. If we take a closer look to *Figure 2.2*, the 3D taxonomy graph, topologies near the bottom must be discarded. This is because a high wire density topology is not good for synthesizing a large adder like a 4098-bit one. Top face topologies have to be preseved then.

Between top face topologies, Brent-Kung must be also discarded because is slower (more logic depth). Finally, Ladner-Fisher has been selected. In fact, it has less area (less fanout) than Sklansky and it is faster than Brent-Kung, so it has a great balance. Once the topology is selected, let us implement it.

Firstly, let us try to parallelize the carries computation by generalizing the generation and propagation functions (G_i and P_i):

$$c_i = G_i + P_i \cdot c_0 = \overline{\overline{G_i} \cdot \overline{P_i \cdot c_0}} \tag{3.3}$$

This expression can be computed with the same *Figure 3.7* circuit, but in parallel. Generalized functions can be obtained by setting a new operation *dot* ($\bullet$) and the recurssive algorithm shown below:

$$(g_j, p_j) \bullet (g_k, p_k) = (g_j + p_j \cdot g_k,\ p_j \cdot p_k) = (\overline{\overline{g_j} \cdot \overline{p_j \cdot g_k}},\ p_j \cdot p_k) \tag{3.4}$$

$$(G_i, P_i) = (g_i, p_i) \bullet (g_{i-1}, p_{i-1}) \bullet ... \bullet (g_1, p_1) \bullet (g_0, p_0) \tag{3.5}$$

As shown, G_i and P_i have linear delay. But, by using Ladner-Fisher's *Dot Procedure* (DP), shown in *Algorithm 3.1*, all G_i and P_i can be computed with a delay of the form:

$$t_{DP} = \log_2(n+2) \cdot t_{dot} \tag{3.6}$$

Algorithm 3.1 : $DP(n, \pi_{in})$

$\pi_{out}(0 : n/2-1) = DP(n/2,\ \pi_{in}(0 : n/2-1))$

$\pi_{med}(0 : n/2-1) = DP(n/2,\ \pi_{in}(n/2 : n-1))$

for $i = 0 : n/2-1$

$\quad \pi_{out}(i + n/2) = \pi_{med}(i) \bullet \pi_{out}(n/2-1)$

end

return π_{out}

Value $\pi_{in}(i)$ correspond to (g_i, p_i). Thus, π_{in} is the complete set of generate and propagate pairs, while π_{out} is the complete set of generalized generate and propagate pairs.

By taking a look back, it can be seen that the carries computation delay is not completely logarithmic. Since (3.3) execution is not included in DP, the carries computation delay has (3.7) form. As a reference value, Brent-Kung's delay form is (3.8) (according with [1] expressions in chapter 11):

$$t_{LF} = \log_2(n+2) \cdot t_{dot} + t_{(3.3)} \tag{3.7}$$

$$t_{BK} = (2\log_2(n+2) - 2)t_{dot} + t_{(3.3)} \tag{3.8}$$

As n is 4096 and t_{dot} is equivalent to $t_{(3.3)}$ (two NAND gates delay), the proposed delay for carries computation against Brent-Kung's one leads to a great difference:

$$\frac{|t_{BK} - t_{LF}|}{t_{BK}} = \frac{|(2\log_2(n+2) - 1) \cdot t_{dot} - (\log_2(n+2) + 1) \cdot t_{dot}|}{(2\log_2(n+2) - 1) \cdot t_{dot}} = \frac{|\log_2(n+2) - 2|}{2\log_2(n+2) - 1} \tag{3.9}$$

$$100 \cdot \left.\frac{|t_{BK} - t_{LF}|}{t_{BK}}\right|_{n=4096} = 43,48\% \tag{3.10}$$

To sum, let us introduce the final proposed PPA algorithm, shown in *Algorithm 3.2*:

Algorithm 3.2

for $i = 0 : n-1$

$p_i = \overline{\overline{a_i} \cdot \overline{b_i}}$

$g_i = a_i \cdot b_i$

end

$\pi_{out} = DP(n, \pi_{in})$

for $i = 1 : n$

$c_i = \overline{\overline{\pi_{out}(i)(1)} \cdot \overline{\pi_{out}(i)(0) \cdot c_0}}$

end

for $i = 0 : n-1$

$s_i = a_i + b_i + c_i \pmod 2$

end

Since Ladner-Fisher topology requires operands with power-of-two length, the 4098-bit adder will be split in two adders of 2 and 4096 bits each.

Finally, the adder/subtractor circuit requires one multiplexor, whose selection signal is c_0, to decide between the normal or the inverted subtrahend. The minuend is always inverted.

3.2 Security Approach

In this section a realistic opponent model is introduced in order to justify the power analysis countermeasures. After that, the countermeasure types already presented in [3] are discussed. Finally, the selected countermeasure is implemented.

3.2.1 Opponent Model

In cryptography, an opponent is understood as a person with technical knowledge and interest in discover the private key. As the method used in this thesis is RSA, the related operation which involves private key is decryption (2.3). Also, as the private key is the exponent in decryption, an SPA attack is very easy to perform.

In fact, DPA is more complex, because it requires high resolution mesurements and a very accurate model. For this reason, in this work DPA countermeasures are not going to be included in the design.

Thus, let us remember why an SPA attack is a good kind of attack to discover the private key in an RSA sheme. As shown in *Algorithm 2.2*, there is a direct dependence between the current bit of exponent (the private key in decryption) and multiplication, or $MP(e_t, \beta_t)$, which is only performed if $\alpha_i = K_i = 1$.

Since there is a power consumption related with multiplication, by looking for extra power consumption in each CLK0 period, the private key can be obtained. This is really easy in the design presented in this chapter, because EN0 literally disables the flip-flops, which causes a low dynamic power consumption.

Let us assume that the circuit specifications are public, for example the clock speed, and it will not be very difficult to detect multiplications in a power consumption trace representation, that can be obtained by mesuring the power supply output with an oscilloscope.

Taking this into account, the countermeasures must hide in some way α when $MP(e_t, \beta_t)$ is computed.

3.2.2 Countermeasure Selection

Following the same order as the presented SPA countermeasures in the state of the art, let us discuss them:

- Performing a multiplication and a doubling in each step independently of the current bit, and later select the result depending on the current bit. This would solve the problem, but the circuit would double the power consumption and the area. So, this is not a good solution.
- Performing the same operation in each cycle. Since multiplication and doubling are actually MP operations, with a clock period of 1 operation (and not 2), but enabling multiplication with a new signal EN1, it should be difficult to differentiate between the two operations. This is the best solution for the proposed design because does not increase the power consumption or the area.
- To set a sequence of operations performed always in the same order is not possible because the number of operations depends on the number of logic ones in the exponent.

Once the convenient countermeasure is selected, let us present the expected power traces for each version, shown in *Figure 3.8*.

The SPA resistant power trace can be obtained by introducing these changes:

- To set a new S0' selection signal which depends on α_i to reach multiplication state. S0' generator is shown in *Figure 3.9*.
- To remove EN0 signal from the flip-flops and to set a new α shift register enabled by EN1 when $\alpha_i = 0$ or when $\alpha_i = 1$ and the doubling operation has been already done. The circuit with these specifications is in *Figure 3.10*.

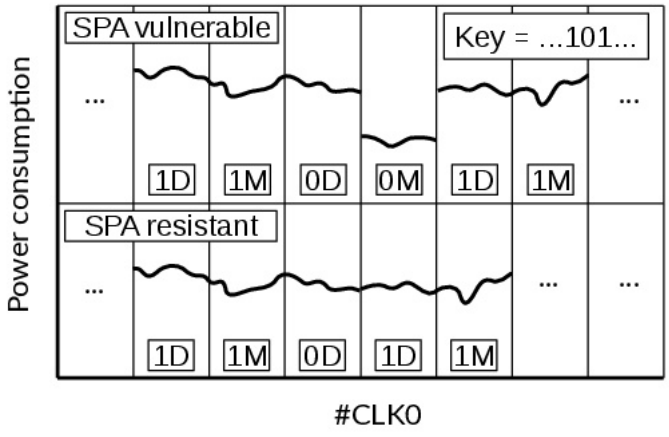

FIGURE 3.8: Estimated power traces

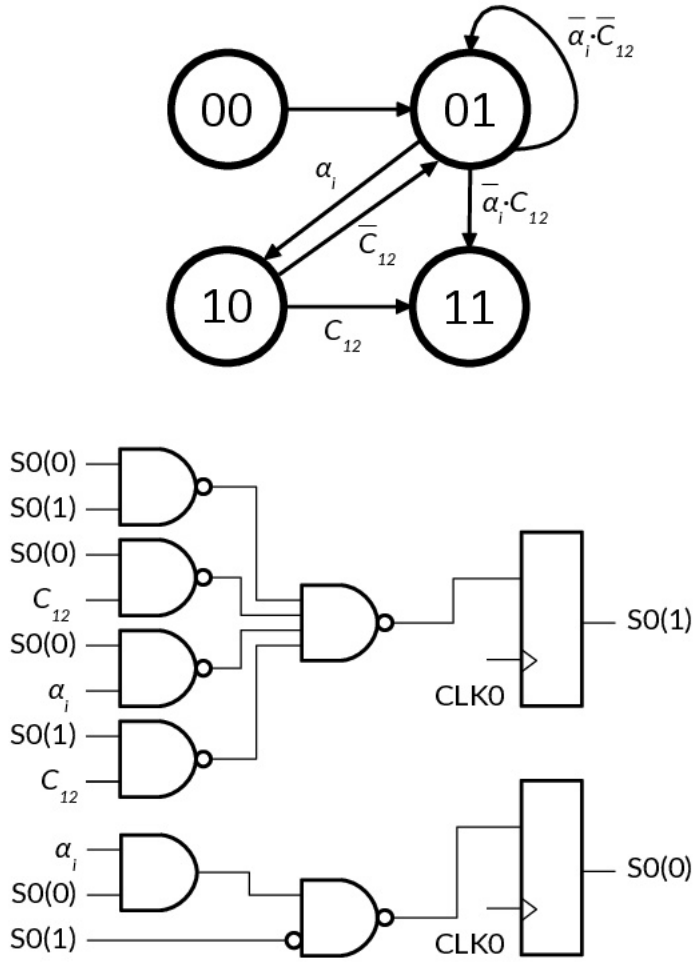

FIGURE 3.9: S0′ signal generator

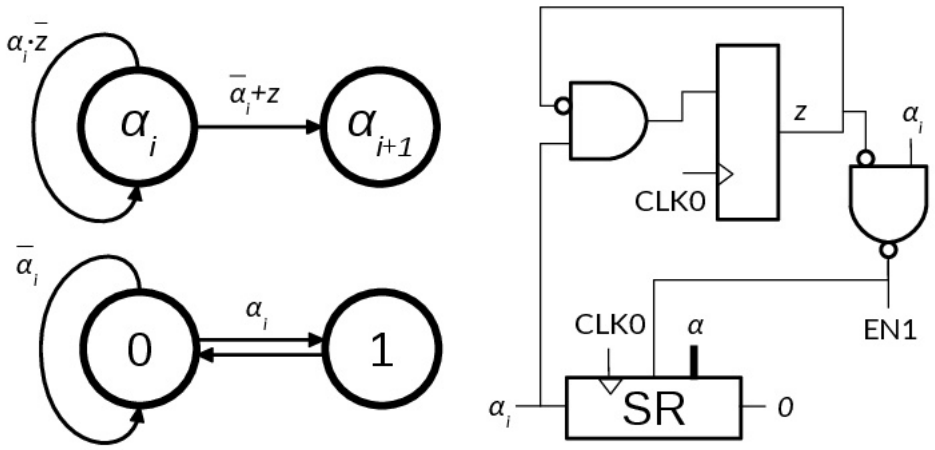

FIGURE 3.10: α_i signal generator

- To add the EN1 signal to the main C_{12} counter, that now operates with CLK0 (not $2 \cdot$ CLK0).

In *Figure 3.10*, an auxiliar logic function z has been introduced to simplify *Moore*'s state machine diagrams. Once the SPA resistant design has been defined, the next chapter shows the presented hypothesis on power traces are certainly accomplished.

Chapter 4

Synthesis and Simulation

This chapter is devoted to the verification of the two designs presented in the last chapter: the original and the SPA resistant versions. At the end of the chapter, simulation results are presented.

4.1 Verification Plan

There are three main goals in the verification plan. Firstly, it has to be proven that both designs work properly from an arithmetic point of view. This means they output the correct results. Secondly, it has also to be proven that SPA resistant power trace hides the private key as expected. Thirdly, the 4096-bit version specifications (area, power and delay) must be obtained.

So, to achieve these goals, the steps below need to be carried out:

- To write a VHDL version of each design. Obviously, this transcription may introduce some changes to the circuits presented in the design chapter. Anyway, these changes do not affect functionality.
- To simulate without gate delays to corroborate that designs are correct from a functional point of view.
- To synthesize the designs into logic gates.
- To simulate with gate delays to prove that designs still work after synthesis. If they do not work, to rewrite the VHDL code to meet the compilator specifications.
- To get the power trace of each version while a modular exponentiation operation is performed.

The flow diagram below shows how these steps are carried out:

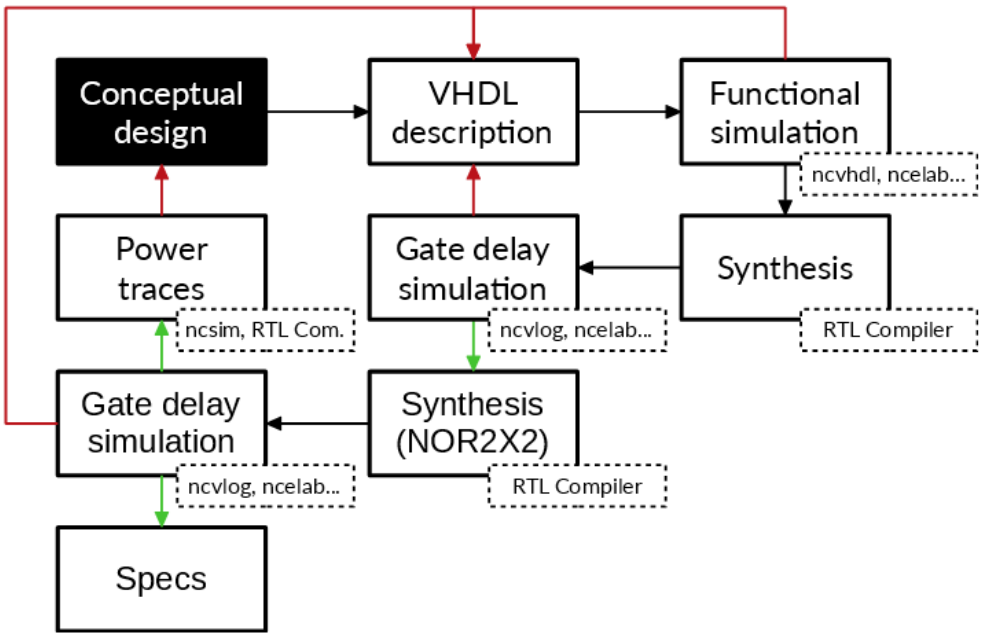

FIGURE 4.1: Verification plan flow diagram

Now, let us explain in detail each step. About the VHDL description, *Appendix A* contains the complete module description using 1993 VHDL version. This description only includes the last version, which is prepared to be synthesized and simulated with delays. In fact, by only simulating without delays the system cannot be considered as fully functional. Let us comment some interesting code fragments.

For example, all selection signals and counters are described in the behaviour domain. Let us see how by seeing the S2 generator and C'_{12} architectures:

```
architecture arch of s2 is ...
  process (CLK2) begin
    if rising_edge(CLK2) then
      if CLK0 = '1' then state <= e0;
      else
        case state is
          when e0 => state <= e1;
          when e1 => state <= e0;
        end case;
      end if;
    end if;
  end process;
  process (state) begin
    case state is
```

```
      when e0 => S2 <= '0';
      when e1 => S2 <= '1';
    end case;
  end process;
end arch;
```

As seen, S2 generator is described as a *Moore*'s state machine. One process controls the states while the other controls the output.

```
architecture arch of c12 is ...
  C12 <= c(log2n) and c(0);
  process (CLK2) begin
    if rising_edge(CLK2) then
      if CLK0 = '1' then c <= (others => '0');
      else
        if CLK1 = '1' then c <= c + '1'; end if;
      end if;
    end if;
  end process;
end arch;
```

In the counter code above, the *unsigned* module of the library is used to add 1 easily to the counter register.

Another interesting architecture is the DP. This is because DP is a recurssive code which generate new structures depending on a generic by using *if...generate* statement:

```
architecture arch of dp is ...
  if_e2: if n = 2 generate
    gen_g(0) <= g(0);
    gen_p(0) <= p(0);
    gen_g(1) <= g(1) or (p(1) and g(0));
    gen_p(1) <= p(1) and p(0);
  end generate;
  if_g2: if n > 3 generate
    ins_dp_low:  entity work.dp
      generic map(n=>n/2)
      port map(g((n/2-1) downto 0), p((n/2-1) downto 0), aux_g_0, aux_p
      _0);
    ins_dp_high: entity work.dp
      generic map(n=>n/2)
```

```
      port map(g((n-1) downto (n/2)), p((n-1) downto (n/2)), aux_g_1, a
      ux_p_1);
    for_high_0: for i in 0 to (n/4-1) generate
      gen_g(i+n/2) <= (aux_p_1(i) and aux_g_0(n/2-1)) or aux_g_1(i);
      gen_p(i+n/2) <= aux_p_1(i) and aux_p_0(n/2-1);
    end generate;
    for_high_1: for i in n/4 to (n/2-1) ...
    gen_g((n/2-1) downto 0) <= aux_g_0;
    gen_p((n/2-1) downto 0) <= aux_p_0;
  end generate;
end arch;
```

As shown, there is a base case where $n = 2$. The other case calls again the DP module. One remarkable comment is that in all the modules the *for...generate* statements are split into 1024 iterations sets. This is because *Encounter RTL Compiler* has an instance generation limit of 1024 in these kind of statementes.

Once the description has been introduced, the commands which allow to simulate are included in *Appendix B*. In a *work* directory a library of VHDL entities and architectures is generated. The next step is synthesis by using the *Encounter RTL Compiler*. The commands needed to carry out this step are also included in the same appendix. Anyway, some comments about these commands are required.

First of all, in order to get easy-to-analyze results, n is going to be set as 16 and not 4096 in order to verify the functionality and the countermeasure. Results are extrapolable to the 4096-bit versions. The library used for the synthesis is from *STMicroelectronics* ®. It is a 1.2V standard 65 nm CMOS library named as CORE65LPSVT_nom_1.20V_25C.

By default, synthesis is performed in a generic way and the compiler decides which gates and how many of them are necessary to take from the library. To increase optimitzation, a flatten design without module hierachy is forced by using:

```
ungroup -flatten -all
```

To get the 4096-bit specifications, more constrains have to be introduced. As it will be shown in results section, there is a problem when overpassing 1024-bit design synthesis, where the *Encounter RTL Compiler* gets stucked. To deal with that, a solution is to make a prediction by using specifications from 4-bit to 1024-bit. In order to get a coherent tendency, compiler automatic optimitzation must be limited.

By using only one type of gate for logic synthesis, this can be accomplished. Since when *Encounter RTL Compiler* is instructed to perform the synthesis using NAND2X2

gates only, it produces a gate level netlist which is not fully functional, in order to get a functional design, NOR2X2 gates have been chosen instead. To achieve this, commands below must be included:

```
set_attribute avoid 1 [find / -libcell *]
set_attribute avoid 0 [find / -libcell *_NOR2X2*]
set_attribute avoid 1 [find / -libcell *_NOR2X25*]
set_attribute avoid 0 [find / -libcell *IVX*]
set_attribute avoid 0 [find / -libcell *DFPQX*]
set_attribute avoid 1 [find / -libcell *SDFPQX*]
```

These commands disable all gates and then enable NOR2X2, inverters of any fanout, and D-type flip-flops. It must be noticed that synthesis output is a Verilog file.

Once the design is synthesized and simulated, power traces have to be generated. For doing that, *Cadence* ® brings a good solution. By setting next commands in *ncsim* console all nodal activity is registered during a period of time:

```
dumptcf -scope ins_modexp -output period_i.tcf -overwrite
run $PERIOD
dumptcf -end
```

Thus, by using a script generator, each period can be monitorized. Later, using the *Encounter RTL Compiler*, the average power in each period can be reported to get a power trace. Finally, after this process, which is long, a script is used to collect all the results and elaborate the power trace for each design. In this case, a period of 100 ns has been used to not interfere in the synthesis optimitzation.

Once the verification plan has been presented, let us see the results.

4.2 Results

The first interesting result is the demonstration that both 16-bit versions work as expected. However, since the simulations are very large, only some parts of them are shown in detail. So, *Figure 4.2*, *Figure 4.3* and *Figure 4.4* show these simulation fragments.

For the simulations shown below, $\alpha = 5DF3$, $\beta = A22A$, $u = AF81$, $2^n \pmod u = 507F$ and $2^{2n} \pmod u = 5F96$. With an online calculator it has been proven that $\beta^\alpha \pmod u = B58$.

Both *Figure 4.2* and *Figure 4.3* show modular exponentiation. It can be seen how EN0 and EN1 work, the C_{12} counter, the change of variable β_t computation at the

beginning and also how S0 changes the state in a different way in each simulation.

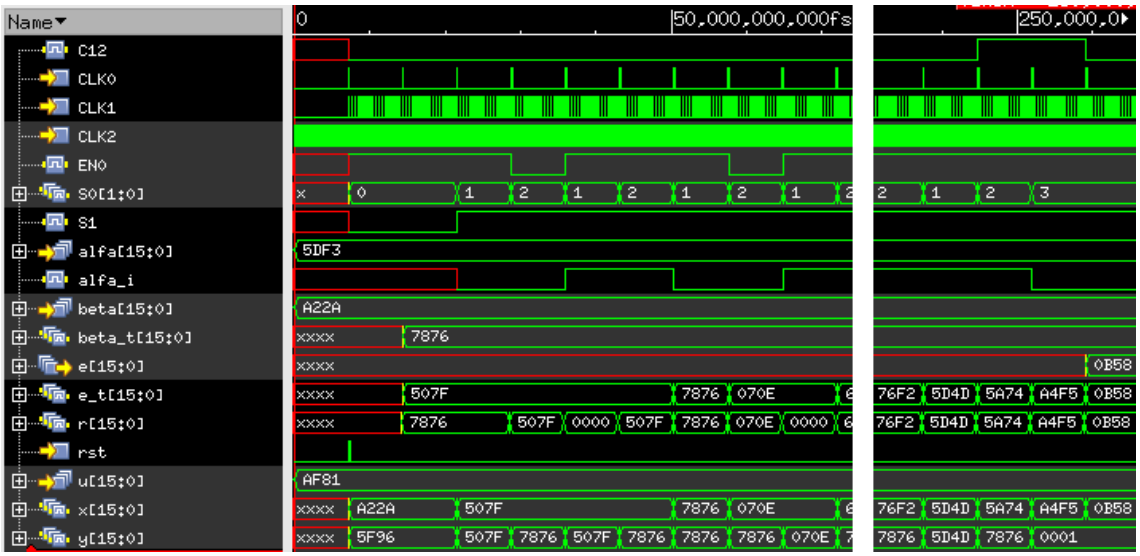

FIGURE 4.2: Modular exponentiation simulation (SPA vulnerable)

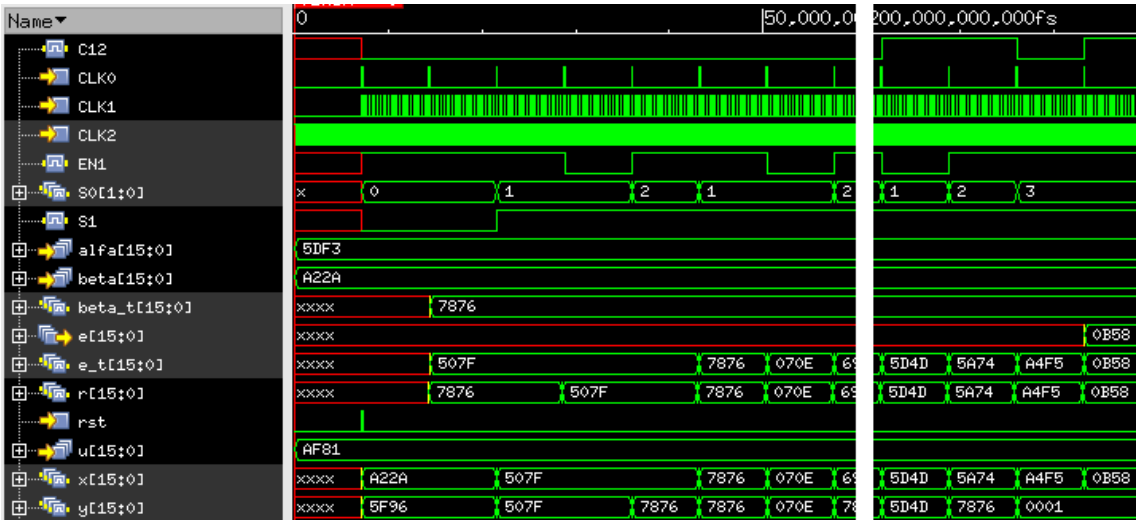

FIGURE 4.3: Modular exponentiation simulation (SPA resistant)

In the next simulation, included in *Figure 4.4*, it can be seen how the MP starts, the clocks shape, the selection signals, the x shift register and, at the end of the MP, the subtraction to obtain the next e_t. Also, some addition operations are shown.

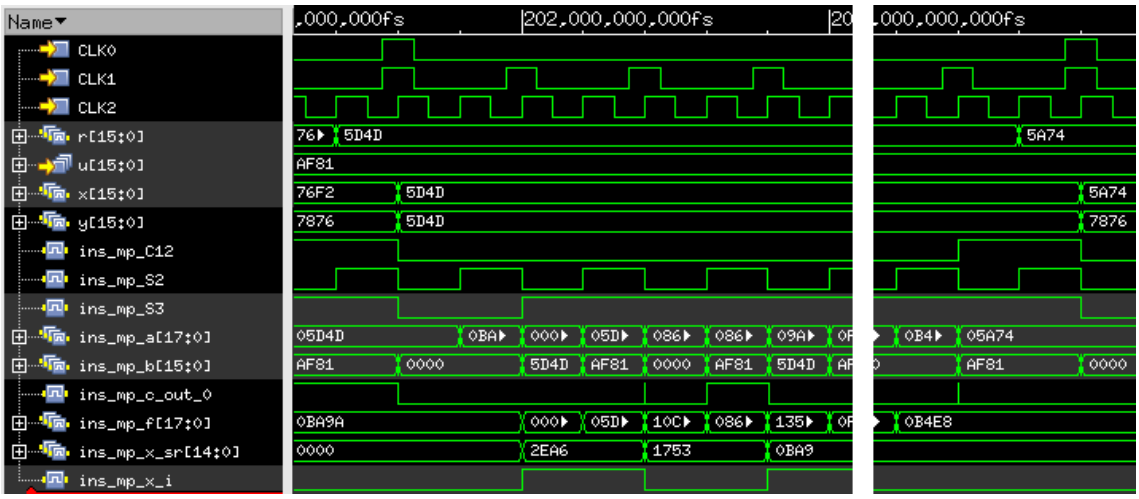

FIGURE 4.4: Montgomery Product simulation

Once simulations have been presented, let us introduce the two power traces, on *Figure 4.5* and *Figure 4.6*, according to the second verification plan goal. Onto these traces the private key in binary form has been printed to allow to see the countermeasure effect.

The periods without a bit correspond to the signal estabilitzation and β_t computation (at the beginning); and the MP inverse to get the result from e_t (last period).

To conclude results presentation, in the next page a table (*Table 4.1*) is provided with the specifications obtained for each version and for each key length from 4 to 1024 bits.

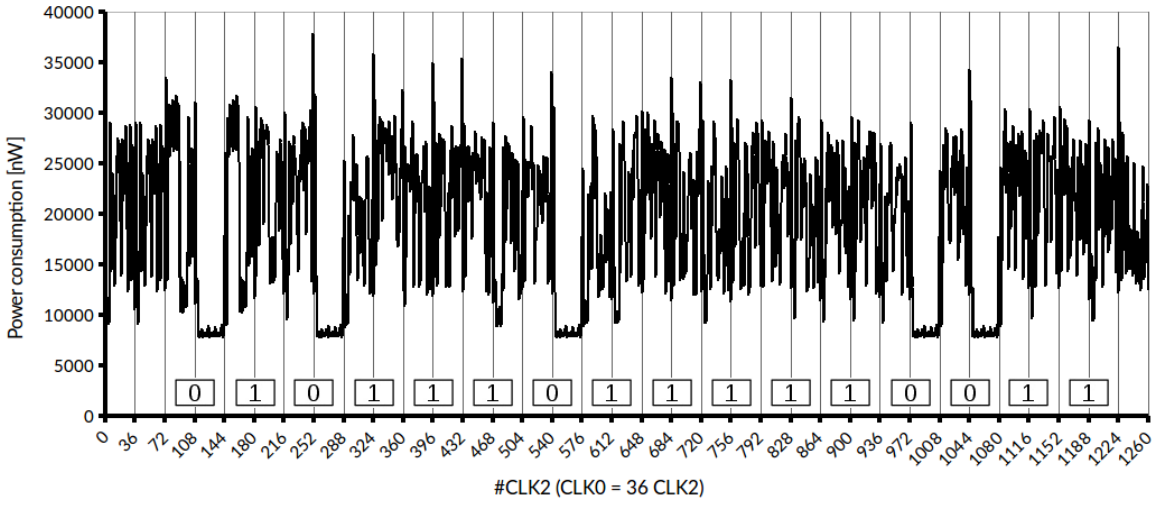

FIGURE 4.5: Power trace (SPA vulnerable)

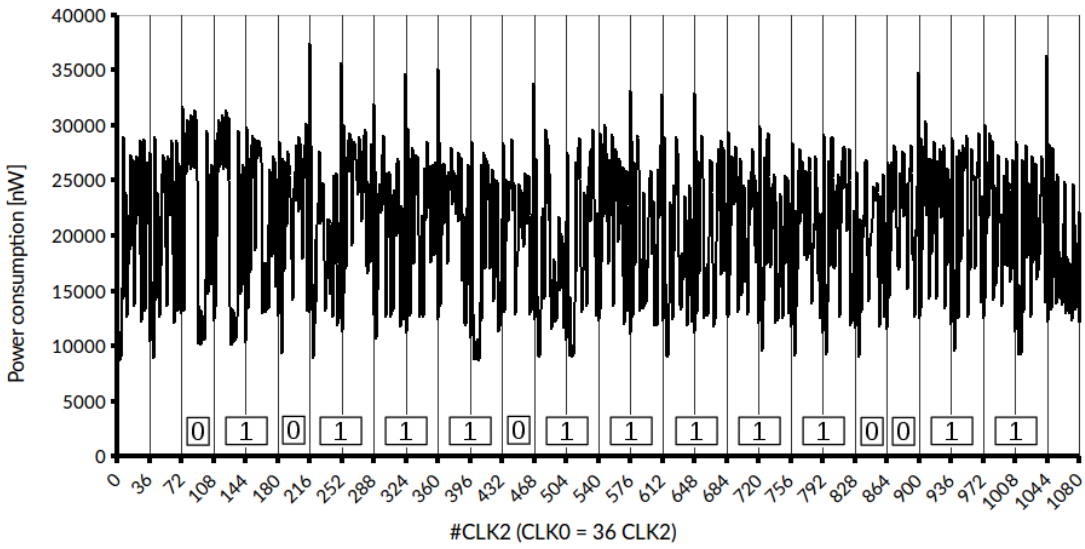

FIGURE 4.6: Power trace (SPA resistant)

SPA	Vulnerable			Resistant		
n	Area [μm^2]	Power [mW]	Delay [ps]	Area [μm^2]	Power [mW]	Delay [ps]
4	1596	0,06	2229	1556 (−2,51%)	0,06 (+ 6,18%)	2214 (−0,67%)
8	2681	0,12	2583	2631 (−1,86%)	0,11 (− 7,85%)	2583 (0,00%)
16	4834	0,22	2940	4781 (+9,06%)	0,19 (−14,40%)	2940 (0,00%)
32	9189	0,39	3274	9139 (−0,54%)	0,29 (−25,86%)	3334 (+1,83%)
64	18009	0,62	3295	17947 (−0,34%)	0,67 (+ 7,79%)	3593 (+9,04%)
128	35872	1,38	3802	35939 (+0,19%)	1,49 (+ 7,63%)	3802 (0,00%)
256	72194	2,87	4775	72133 (−0,08%)	2,85 (− 0,75%)	4774 (−0,02%)
512	146227	6,44	5997	146151 (−0,05%)	4,84 (−24,74%)	6035 (+0,63%)
1024	296232	11,34	6922	296066 (−0,06%)	11,80 (− 4,06%)	6950 (+0,40%)
Estimated specifications						
2048	590784	23,17	8020	590511 (−0,05%)	22,90 (− 1,20%)	8067 (+0,59%)
4096	1181919	46,31	9078	1181400 (−0,04%)	45,86 (− 0,97%)	9137 (+0,65%)

TABLE 4.1: Circuit specifications summary (NOR2X2)

Using this data, an estimation for a 4096-bit design has been carried out. The power specifications have been obtained in nW, but they are shown in mW.

The estimated specifications in the table above have been obtained from the linear regressions in (*Figure 4.7* and *Figure 4.8*) and from a logarithmic regression in (*Figure 4.9*). The logarithmic regression is performed as a linear one with a $log_2\ n$ change

of variable, taking the values after 64-bit. Trend lines are included on *Table 4.2*. Delay refeers to the critical path, which is the addition operation in each CLK2 period.

Each specification in the SPA resistant version column includes a relative comparison to the SPA vulnerable version.

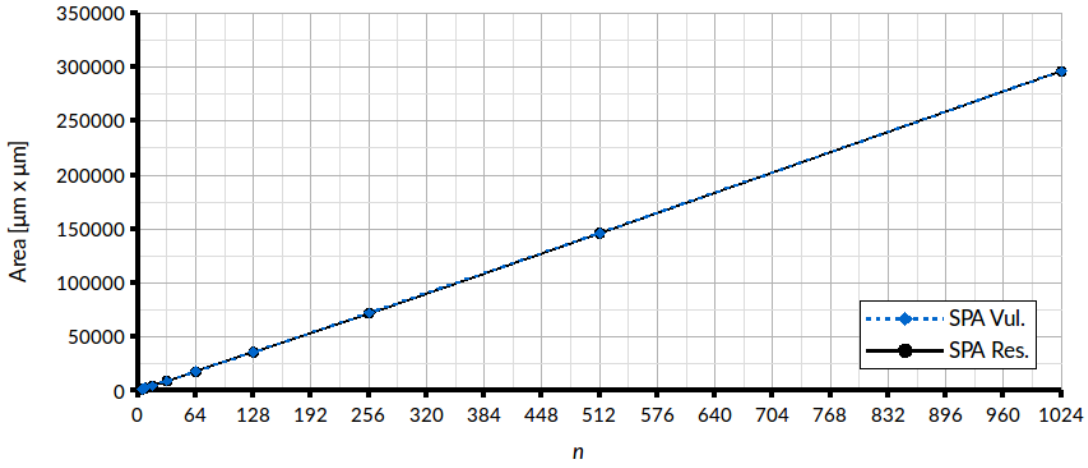

FIGURE 4.7: Area - n graph

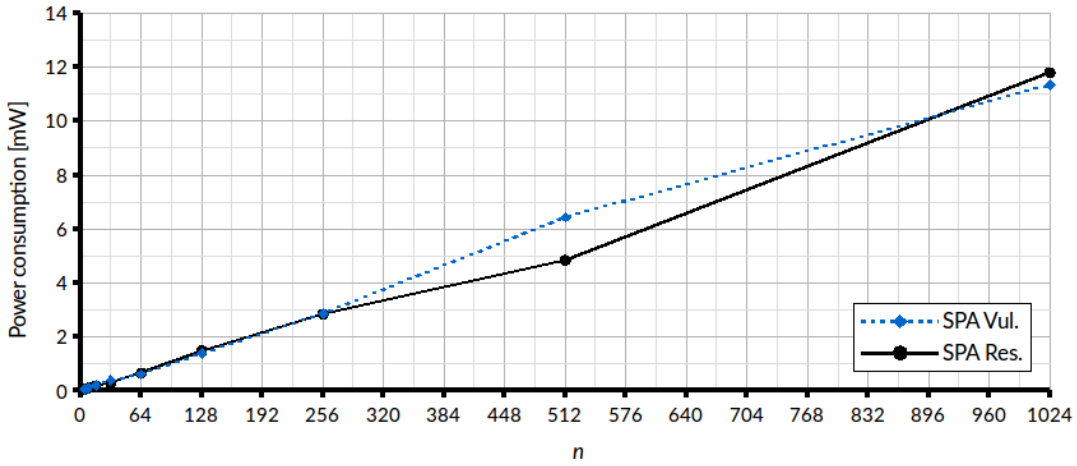

FIGURE 4.8: Power consumption - n graph

SPA	Vulnerable $(R^2 \geq 0,99)$	Resistant $(R^2 \geq 0,99)$
Area [μm^2]	$288,64n - 350,30$	$288,52n - 377,76$
Power [nW]	$11297,07n + 38412,21$	$11213,30n - 69061$
Delay [ps]	$1058,2 \log_2 n - 3620,7 \ \ [n \geq 128]$	$1070,5 \log_2 n - 3709 \ \ [n \geq 128]$

TABLE 4.2: Trend lines for area, delay and power on both designed versions

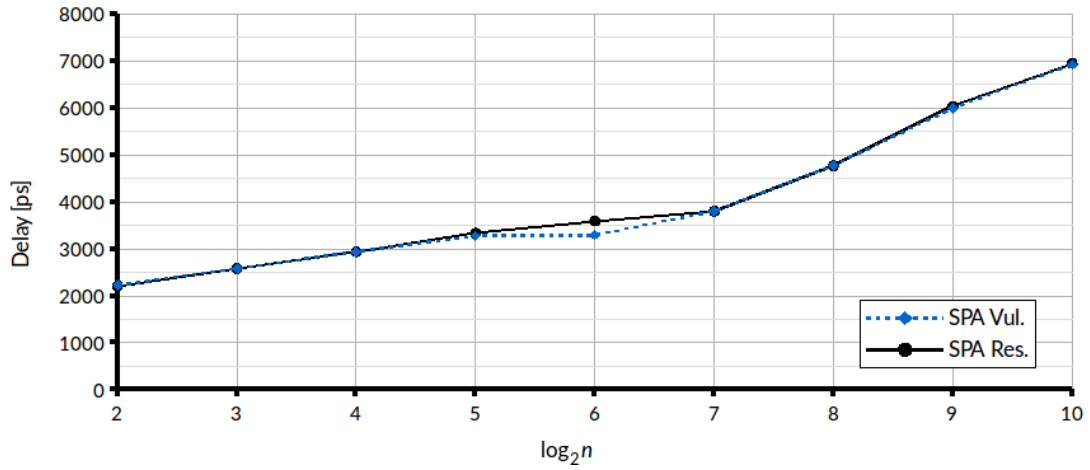

FIGURE 4.9: Delay - $\log_2 n$ graph

Chapter 5

Discussion

5.1 Comments on Results

In this section results are discussed in the same order as they have been presented in the last chapter. Thus, let us start with simulation including delays. As seen in *Figure 4.2* and *Figure 4.3*, both versions work properly. Details have not been given, but both designs have been evaluated with various inputs in order to corroborate their functionality.

These simulations show that there is an initial period to allow signals estabilization. Such estabilization time has not been mentioned in *Chapter 3* but it is mandatory in order to get the system ready for the computation.

The execution time is clearly different between versions. In the case shown, there is a 14,3 % time reduction with the SPA resistant version. If the key bits are balanced properly, this time reduction could rise until the 50 % mark.

Without a power trace, in these simulations is also noticeable that the first version is SPA vulnerable. That is because when $\alpha_i = 0$ all values are mantained during some CLK0 periods.

Another remarkable fact is about the clocks duty cycles and their timings. In *Figure 4.4* is shown how CLK0 and CLK1 must be advanced to ensure they enable properly when CLK2 rises. Also duty cycles are small to avoid interferences.

In this simulation can be seen how in each MP there is also one stabilitzation CLK1 period at the beginning. After that, f become a zeros vector. In each CLK2 period an addition is performed. When S2 = 1, if a is even, then the addition is not computed, because a can already be right-shifted.

As a last comment of these simulations, subtraction performed in the MP shown above corresponds to the case where $r_n < u$, thus, e_t gets r_n.

Now, let us comment power traces. In fact, there are not many surprises because they meet with hypothesis in *Figure 3.8*. If the opponent can generate a power trace like the one in *Figure 4.5*, only by setting zeros on low power periods and ones in the rest, the private key is revealed. In opposition to the SPA vulnerable version, the SPA resistant version manages to hide these low power gaps. The rest of the power traces are identical.

Finally, let us comment on specifications. As expected, area increases linearly (*Figure 4.7*). Ladner-Fisher topology doubles the number of DP entities when the operand length is doubled, so, that is proportional. All the other circuit parts seem to follow the same proportional rule.

If the area tendency is linear, power consumption is expected to be linear too. This is shown in *Figure 4.8*, with one exception: 512-bits. Since R^2 is 0,99; maybe that exception is because the compiler optimizes this version in a different way, or it is just specifications variance.

Delay tendency, shown in *Figure 4.9*, denotes one important thing about the compiler: after 128-bit, the adder optimitzation changes.It is known that the *Encounter RTL Compiler* algorithms identify the adder structures. Maybe after 128-bit it can not identify properly these structures or any other applied optimization after this point (128 bits) is not capable of maintaining the previous delay tendency. This explains a rising of delay per bit.

In fact, all values except 64-bit SPA vulnerable version (which does not fit as well as expected), correspond to one of the two logarithmic regressions that can be made, according to Ladner-Fisher's carries delay form. The one after 128 bits has been already presented (4.1), and the one before 128 bits corresponds to $350,9 \log_2 n + 1529,2$; in the SPA resistant version.

There are no relevant differences between area or delay specifications for both versions (<10%). However, in the SPA resistant version, in average, there is a decrease of power consumption.

Thus, if the compiler does not change optimitzation algorithm after 128 bits, the estimation introduced will be close enough from the real result. Anyway, it can be almost ensured that CLK2 can operate at 100 MHz in a 4096-bit version. If the private key has half zeros, modular exponentiation would be computed in about 504 ms, which is good enough. This is deduced from:

$$\#CLK0 = 3 + 2 \cdot \#(\alpha_i = 1) + \#(\alpha_i = 0) \tag{5.1}$$

$$\#CLK1 = (2 + n) \cdot \#CLK0 \tag{5.2}$$

$$\#CLK2 = 2 \cdot \#CLK1 \tag{5.3}$$

$$t_{\beta^{\alpha} \pmod{u}} = t_{CLK2} \cdot \#CLK2 \tag{5.4}$$

This value of 504 ms, is the average execution time. The fastest scenario happens with a full zeros private key, then, modular exponentiation is computed in about 336 ms. The slowest scenario happens with a full ones private key, performing modular exponentiation in about 672 ns, which is the execution time of all the SPA vulnerable runs independently of the private key.

5.2 Estimated Costs

The next table represents a summary of the costs of the equipment and *Non-Recurring Engineering* (NRE) necessary to obtain the thesis results.

Item	Time [h]	€/h	Total [€]
NRE			
• Design	200	15	3000
• Synthesis	40	15	600
• Verification	80	15	1200
• Director's supervision	80	40	3200
Equipment			
• Server amortization	80	0,130	10
• Server usage	80	0,300	24
• Computer amortization	320	0,022	7
• Computer usage	280	0,036	10
• *Cadence* ® license	80	0,205	16
		Total [€]	8067

TABLE 5.1: Estimated costs summary

The cost per hour of NRE is for a junior engineer with a salary of 30000 €/year. The server and computer usages have been calculated with a price of 0,12 €/kWh and powers of 2,5 kW (server and cooling) and 0,3 kW (computer).

The amortization has been calculated by using a server cost of 8000€ and a computer cost of 950€ in 7 years (server) and 5 years (computer). The license price can be found in the *Cadence* ® webpage: 1800€ in the first year.

5.3 Conclusions

As derived from the result discussion above, all the goals have been successfully fulfilled. Both designs are working in the way concieved in the design chapter. Also, the countermeasure has efficiently managed to hide the private key to the opponent by making the system SPA resistant.

Although there have been lots of practical difficulties while using *Cadence* ® software, results, including NOR2X2 synthesis, are accurate enough to get some operating bounds, such as the average execution time of 504 ms.

Since the design presented works in a sequential scheme, its area specifications should be lower than most 4096-bit versions of an RSA system, provided that their adders/subtractors were fully combinational.

Once the 16-bit design of the SPA resistant version has been proven to work and before to think about synthesizing a definitive 4096-bit version, it would be interesting to explore other side-channel attack conuntermeasures to prevent other kind of attacks that the one shown in this thesis. This means to redesign and to ensure that the countermeasures do not interfere between them.

Surely, the most important concept introduced in this thesis is that just modifying the control system (not the data system), a modular exponentiation module can resist side-channel attacks while reducing overall execution time.

As personal considerations, it has been interesting to explore PPA adder topologies, and more generally, all the whole set of algorithms needed to complete a modular exponentiation module. It has been motivating to work on a large and complex hardware design, which has application nowadays.

Bibliography

[1] J.-P. Deschamps, G. J. A. Bioul, and G. D. Sutter, *Synthesis of Arithmetic Circuits: FPGA, ASIC, and Embedded Systems*, Wiley, Ed. 2006, ISBN: 978-0-471-68783-2.

[2] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, vol. 21 (2), pp. 120–126, 1978.

[3] A. Das, "Differential scan-based side-channel attacks and countermeasures", Doctoral Thesis, KU Leuven, Arenberg Doctoral School, 2013.

[4] Y.-T. Pai and Y.-K. Chen, "The fastest carry lookahead adder", *Proceedings of the Second IEEE International Workshop on Electronic Design, Test and Applications*, 2004.

[5] J. Miao and S. Li, "A novel implementation of 4-bit carry look-ahead adder", *International Conference on Electron Devices and Solid-State Circuits*, 2017.

[6] A. Lloris, E. Castillo, L. Parrilla, and A. García, *Algebraic Circuits*, Springer, Ed. 2014, ISBN: 978-3-642-54648-8.

[7] D. Harris, "A taxonomy of parallel prefix networks", *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*, 2003.

[8] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design: A circuits and systems perspective*, Fourth Edition, Pearson, Ed. 2010, ISBN: 978-0-321-54774-3.

[9] E. Barker, "Recommendation for key management part 1", *NIST Special Publication 800-57 Part 1*, 2016.